Use **www.w3schools.com** for this lab.

For Bootstrap use version 3.

Feel free to find other resources to help as well.

**Exercise 1**

[**X**] Start with the MVC application from the second lab (week 3/in memory database) or third lab (week 4/real database) by copying it to another folder.

[**X**] Create an MVC Controller for a "Bootstrap" entity with an empty view for the "Index" action.

[**X**] Add a link in the navigation bar to the "Bootstrap" index page.  
.

[**X**] Copy the Bootstrap example HTML from the w3schools **Home** section into the new page.

[**X**] Run the application, click the "Bootstrap" navigation link you added and view page.

[**X**] Add content to the page describing the following (place your answers on the page using the layout provided or add to it if you wish).

Elements described on page

* 1. Describe the page layout you see.
  2. Describe what happens as you resize the page.
  3. Using w3schools**Getting Started**and **Grid Basic**sections, discover what the CSS classes on the page are doing and describe there use.

1. In the **Getting Started** section, read about the DOCTYPE and the meta tags.  No need to add anything for this to your page.

DOCTYPE = describes the type a document / language of document the file is

META = is a tag that helps render the characters on the page properly.

**Exercise 2 (continue using the new Bootstrap page)**

[**X**] Using the w3schools **Typography**section, highlight a segment of text as if you were using a yellow highlighter.

[**X**] Highlight a segment of the text in the contextual color representing danger.  To accomplish this, you can surround a portion of text with the <span> tag.

[**X**] Add two more typography styles to the page content.

**Exercise 3 (use one of your application's Index pages that displays a table of one of your entities (e.g. Person).**

ALL EDITS DONE TO VIEWS/USER/LIST

[**X**] Make the table striped and with a border.

[**X**] Add a Bootstrap "Well" to your page to display a heading for the page.

[**X**] Add an alert that can be dismissed by the user to the page when the count of entities being displayed surpasses 2.  Look at the w3schools.com **Alerts** section to find how to create an alert.  (Hint: You'll need to add some Razor syntax to conditionally display the alert.  The collection model has a "Count" property)

[**X**] Make all of the links on you page buttons.  Make the create link a large button and the row actions (edit, details, delete) extra small buttons.  See the w3schools.com **Buttons** section for more details.  
Here is the HTML helper syntax you'll need:  
  
                @Html.ActionLink("button label", "controller action", new { id = item.Id }, new {@class = "Bootstrap classes for button"})

**Exercise 4 (Continue using the page you chose in Exercise 3)**

[**X**] Add a script section at the bottom of the entity list view like this:  
  
@section scripts {  
     <script>  
     </script>  
}  
  
This is where you will put the JavaScript that you'll add to this page.

[**X**] Find the following at the end of the common layout page:  
  
@RenderSection("scripts", false);  
  
This causes the MVC framework to add the script section you specified in step 1 to the resulting rendered HTML page along with the other scripts.

[**X**] Add a jQuery document ready function to the script section in step 1.  This is described in the w3schools.com jQuery Syntax section.  
You can find an example of this in the Week 7 example on the class GitHub repo.

[**X**] Inside the ready function add the following line of JavaScript code:  
  
alert("Hello");  
  
When you run the application and go to the page you should see an alert dialog displayed with your message.  If you don't see the alert dialog then click F12 to open the browser's developer tools.  Go to the console display to see if there was an error.  Let me know if you need help getting this to work.  
  
After you've got the alert to display, remove the call to the alert.

**Exercise 5 (Continue using the page you chose in Exercise 3)**

1. Add two buttons underneath the table.  One to show the table and one to hide the table.  You can use the <button> tag to create the buttons.
2. Style the buttons with Bootstrap button styling.  You should see the buttons highlight as you hove over them.
3. Give each button a unique ID so that you can reference them with jQuery.  You might want to review how jQuery selectors work by reading the w3schools.com **jQuery Selectors** section.
4. Add two click event handlers to the jQuery ready function (where you previously had the call to the alert function).  One for the show table button and the other for the hide table function.  If you're not sure how to add event handlers then read the w3schools.com **jQuery Events** section.
5. Add a call to alert in each event handler function to test much like you tested the document ready function exercise 4.
6. Once you've successfully tested that your event handler are working then replace the calls to alert with functions that hide and show the table.  Remember, you'll need to add a unique id to the table now in order to reference it from jQuery.  Read the w3schools.com **jQuery Effects** section to see how to hide and show.  You might want to play around with the fade and animate as well.

**Exercise 6** **(Extra Credit)**

1. Make it so that each row of the table highlights when you pass the cursor over a row.  Go back to the **Bootstrap Table** section and find the class you need to add.
2. Add a click handler for the rows of the table.  Hint: You'll need to add a class to the rows created by MVC Razor and then attach a click handler to each row using a jQuery class selector.
3. You can use the "alert" test you used before to check your click handler.
4. Add a class to one of the table cells (<td> tag) so that you can select that particular piece of information from the row.
5. Add the following code to your row click handler:  
     
   var value = $(this).find('.cell-class-name-you-added').text().trim();  
     
   What is this doing?  The "this" keyword is saying you are referencing the element the click was handled for (the row).  The "find" function says to find the element inside of the "clicked" element that has a certain class.  Once you have that element you can get it's contents using the text() function.  The trim() simply removes extra white space.
6. Once again you can use an alert() function to test if you are actually getting the correct value.
7. Now add a <div> element to the page with a unique id.
8. Display the value from the row you clicked inside of the <div> tag you added in step 7.  You can use the .text(value) to set the value.